

EXHIBIT C

5.0 Reference

This section describes the actual OLE Interfaces exposed, the definitions of the data structures used when passing data around, and the definitions of each class used internally by the XMC Motion Control Component.

5.1 Interfaces

Other than the standard OLE interfaces IUnknown, IClassFactory, and IDispatch, and the custom XMCAPI OLE interfaces, there are no special interfaces exposed by the XMC Motion Control Component. The diagram below displays a graphical representation of the interfaces exposed by the XMC Motion Control Component.

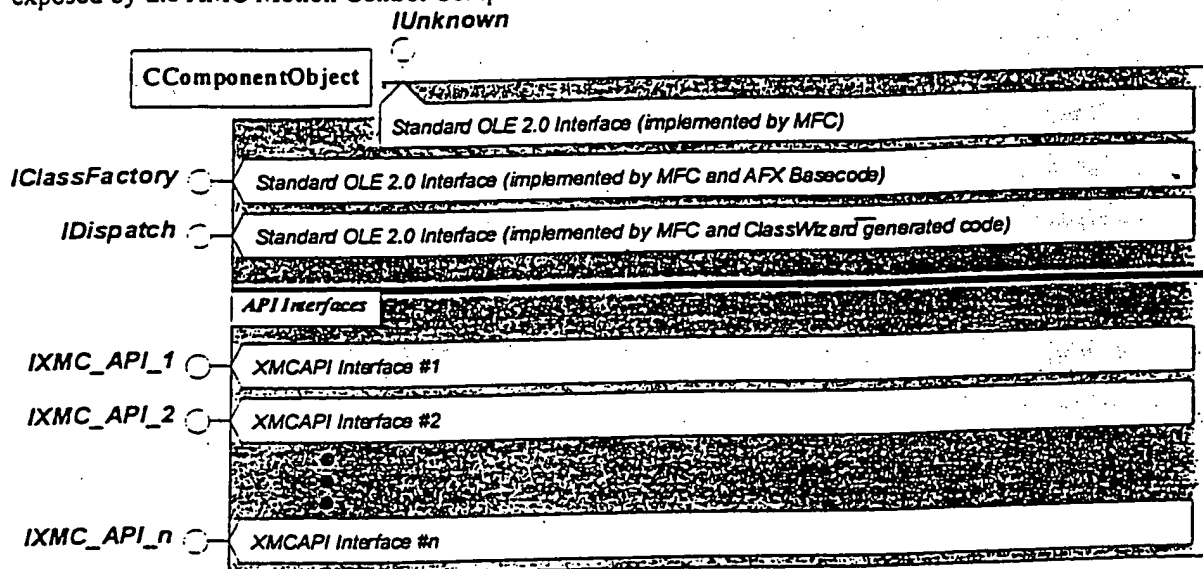


Figure 8 Interface-Map.

For more information on the standard OLE interfaces see the *OLE 2 Programmer's Reference*. And, for more information on the custom XMCAPI interfaces, see the *XMCAPI Reference*.

5.2 Exported Functions

The following are the functions exported by the driver DLL.

XMC_COMPONENT_MODULETYPE	DLLGetModuleType(void);
LPCLSID	DLLGetCLSID(void);
BOOL	DLLRegisterServer(void);
BOOL	DLLUnRegisterServer(void);

5.3 Structures and Defines

This section defines all structures, enumerations, and defines used by the driver.

5.3.1 XMC_COMPONENT_MODULETYPE Enumeration

The following enumeration is used to mark the module type of the component.

```

enum XMC_COMPONENT_MODULETYPE
{
    XMC_COMPONENT_MT      = 0x7000
};
  
```

5.3.2 XMC_API Enumeration

The following enumeration is used to index all XMC API supported by the component.

```
enum XMC_API
{
    XMC_API_MOTION_MOVEABS,
    XMC_API_MOTION_MOVEREL,
    XMC_API_MOTION_KILL,
};
```

5.3.3 XMC_MAPMODE Enumeration

The following enumeration is used to specify the type of mapping to use when converting between the MCS and PCS.

```
enum XMC_MAPMODE
{
    XMC_MM_METRIC,
    XMC_MM_ENGLISH,
    XMC_MM_NATIVE
};
```

5.4 Classes

This section contains the definition of all classes used by the XMC Motion Control Component in its implementation.

5.4.1 CCmpntDisp Class

The CCmpntDisp class acts as the dispatch, or control, object used to separate all OLE details from the code implementing the driver internals. This object coordinates all operations taking place in the driver by directing other C++ object to carry out the work of the operation. The following is the definition of the CCmpntDisp class.

```
class CCmpntDisp
{
public:
    //---- Constructors & Destructors ----
    CCmpntDisp( void );
    ~CCmpntDisp( void );

    //---- Initialization ----
    DWORD Initialize( void );

    //---- Actions ----
    DWORD FireAPI( XMC_API apiIDX, ... );

private:
    //---- Private Methods ----
    DWORD verifyParams( ... );
    DWORD verifyState( ... );

    //---- Private API Methods ----
};
```

```

//---- Private Data ----
CDriverMgr  m_drvMgr;
};

```

5.4.2 CDriverAdmin Class

The CDriverAdmin class is used to directly communicate with the driver administrator component. All OLE related details are encapsulated within this class. The following is the definition of the CDriverAdmin class.

```

class CDriverAdmin
{
public:
    //---- Constructors & Destructors ----
    CDriverAdmin( void );
    ~CDriverAdmin( void );

    //---- Initialization ----
    DWORD Initialize( void );

    //---- Actions ----
    DWORD EnumDriver( LPENUMDRIVER *ppEnumDriver );

    DWORD RegisterDriver( LPCTSTR pszFileName,
                          BOOL bEnabled,
                          LPXMC_HDRIVER phDrv );
    DWORD RegisterStream( LPCTSTR pszFileName,
                          BOOL bEnabled,
                          XMC_HDRIVER hDrv,
                          LPXMC_HSTREAM phStrm );
    DWORD UnRegister( XMC_HDRIVER hDrv, XMC_HSTREAM hStrm );
    DWORD GetSupportInfo( LPXMC_DRIVERINTERFACESUPPORTINFO rgDIFS );

    //---- Debugging ----
    DWORD EnableDiagnostics( BOOL bEnable );

private:
    //---- Private Data ----
    LPXMC_DRIVERADMIN m_lpDA;
    LPXMC_DRIVERADMINDEBUG m_lpDADbg;
};

```

5.4.3 CDriverMgr Class

The CDriverMgr class manages all active drivers used in the XMC system. Currently, XMC only supports single driver operation. When the software model is expanded to support multiple driver, the CDriverMgr will be responsible for managing all drivers by virtualizing them to form one contiguous set of axes. The CDriverMgr also controls the process of unit mapping. Below, is the definition of the CDriverMgr class.

```
class CDriverMgr
{
public:
    //---- Constructors & Destructors ----

    CDriverMgr( void );
    ~CDriverMgr( void );

    //---- Initialization ----

    DWORD Initialize( void );

    //---- Setup Operations ----

    DWORD RegisterDrivers( LPENUMDRIVER *ppEnumDriver,
                          LPENUMINTERFACESUPPORT *ppIFS );
    DWORD SetMappingMode( XMC_MAPMODE mapMode );
    DWORD SetBaseUnits( double dfMCPPerMM, double dfECPPerMM );

    //---- Actions ----

    DWORD FireAPI( XMC_API apiCMD, ... );

private:
    //---- Private Methods ----

    DWORD mapPCStoMCS( ... );
    DWORD mapMCStoPCS( ... );

    //---- Private Data ----

    CUnitMapper          m_unitMapper;
    CDriver              *m_rgDrivers;
    DWORD                m_cbDrivers;
};
```

5.4.4 CUnitMapper Class

The CUnitMapper class is used to map all measurements between MCS and PCS. The CUnitMapper class is defined as follows.

```
class CUnitMapper
{
public:
    //---- Constructors & Destructors ----

    CUnitMapper( void );
    ~CUnitMapper( void );

    //---- Initialization ----
```

```

DWORD SetMappingMode( XMC_MAPMODE mapMode );
DWORD SetBaseMCUnit( double dfMCPerMM );
DWORD SetBaseECUnit( double dfECPerMM );

//---- Actions ----

double MapMCStoPCS( double dfVal );
double MapPCStoMCS( double dfVal );

private:
//---- Private Data ----

double      m_dfMCStoPCS;
double      m_dfPCStoMCS;
double      m_dfMCPerMM;
double      m_dfECPerMM;
};

```

5.4.5 CDriver Class

The CDriver implements the internal XMCSPi table that stores pointers to exposed methods in both the XMC Driver and XMC Driver Stub. A function pointer pointing to *each extended* XMCSPi function, not supported by the XMC Driver that may be simulated through a software algorithm, is stored in the XMCSPi table. All other entries, in the table, consist of function pointers to XMCSPi functions implemented by the XMC Driver. Graphically, the internal data map for the CDriver class is as follows.

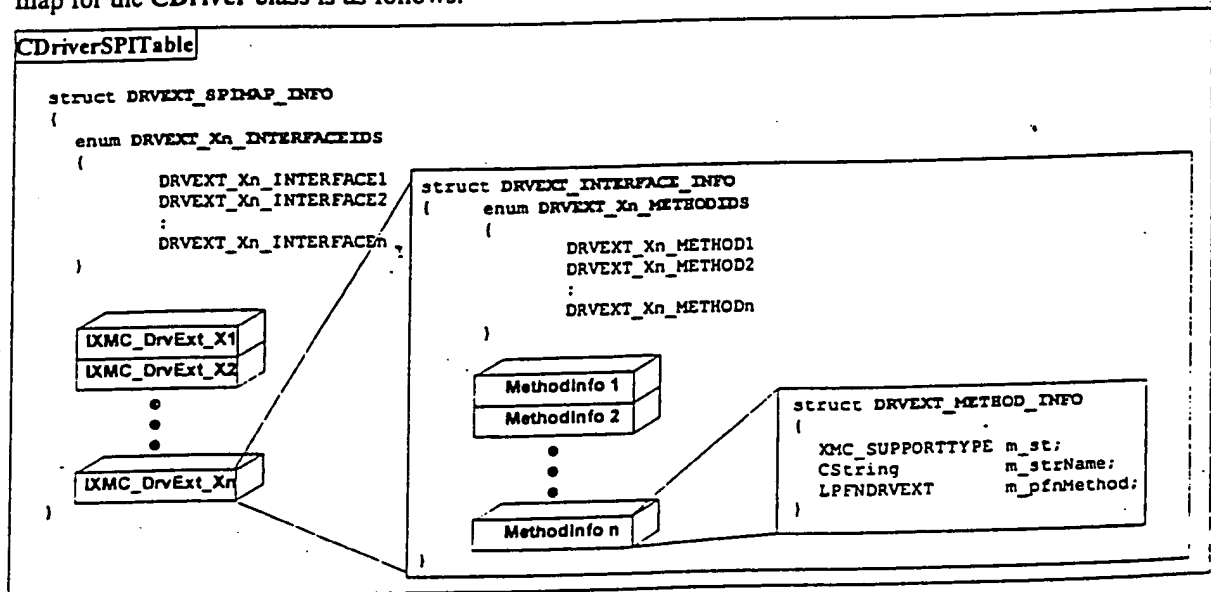


Figure 9 Data-Map - CDriver class with XMCSPi table.

The purpose of the CDriver class is to manage all connections to both the driver and driver stub. All OLE specific details are encapsulated within the CDriver class. The following is the definition of the CDriver class.

```

class CDriver
{
public:
//---- Constructors & Destructors ----

CDriver( void );
~CDriver( void );

```

```
//---- Initialization ----

DWORD Initialize( void );
DWORD Initialize( LPUNKNOWN pDrvUnk, LPUNKNOWN pDrvStbUnk );

//---- Actions ----

DWORD Attach( LPUNKNOWN pDrvUnk, LPUNKNOWN pDrvStbUnk );
DWORD Detach( LPUNKNOWN *ppDrvUnk, LPUNKNOWN *ppDrvStbUnk );
DWORD FireCoreSPI( XMC_CORESPI corespID, ... );
DWORD FireExtSPI( XMC_EXTSPI extspiID, ... );

private:
//---- Private Data ----

DRVCORE_SPIMAP_INFO      m_rgCoreSPIMap[];
DRVEXT_SPIMAP_INFO      m_rgExtSPIMap[];
LPUNKNOWN               m_pDrvUnk;
LPUNKNOWN               m_pDrvStbUnk;
};
```

NOTE: For more information on the *XMC_CORESPI* and *XMC_EXTSPI* types see the *XMC Motion Control - Driver Design Guide*.